From b.carlson@compmail.com Fri Dec 3 17:49:32 1993
Received: from data.nas.nasa.gov by wk45.nas.nasa.gov (5.67-NAS.6/NAS.3-sgi)
id AA02061; Fri, 3 Dec 93 17:49:32 -0800
Received: from sprintf.merit.edu by data.nas.nasa.gov (5.67-NAS.6/NAS.3-mx)
id AA04609; Fri, 3 Dec 93 17:49:07 -0800
Received: by sprintf.merit.edu (5.64/1123-1.0-X.500)
id AA10154; Fri, 3 Dec 93 20:49:54 -0500

527108

From: b.carlson@compmail.com
Received: by sprint.com (SXG 7.0a/sprintf2.0) with X.400
id 00gzymzua001; 4 Dec 93 01:49:32 UT
Date: 3 Dec 93 20:07:29-0500
P1-Message-Id: US*TELEMAIL;AGJD-6005-3621/08
To: simon@nas.nasa.gov
Subject: Final Gordon Bell article
Message-Id:
<"AGJD-6005-3621/08"*/G=B/S=CARLSON/O=COMPMAIL/ADMD=TELEMAIL/C=US/@sprin
Status: R

Alan Karp said you needed a copy of the final Gordon Bell judges' summary. Here it is. Let me know if you'd prefer a faxed version... or if the long e-mail version doesn't transmit properly.

Bob Carlson
Computer Magazine
(714) 821-8380

JUDGES' SUMMARY

Title

1993 Gordon Bell Prize Winners

Authors

Alan H. Karp, Hewlett-Packard Laboratories

Don Heller, Rice University

Horst Simon, Computer Sciences Corporation

Abstract

Records were shattered and the number of entries nearly doubled in this year's Gordon Bell Prize competition. The winners set marks almost five times higher than the previous performance record.

The Gordon Bell Prize recognizes significant achievements in the application of supercomputers to scientific and engineering problems. In 1993, finalists were named for work in three categories:

- Performance, which recognizes those who solved a real problem in the quickest elapsed time.
- Price/performance, which encourages the development of cost-

effective supercomputing.

- Compiler-generated speedup, which measures how well compiler writers are facilitating the programming of parallel processors.

The winners were announced November 17 at the Supercomputing 93 conference in Portland, Oregon. Gordon Bell, an independent consultant in Los Altos, California, is sponsoring $2,000 in prizes each year for 10 years to promote practical parallel processing research. This is the sixth year of the prize, which Computer administers.

Something unprecedented in Gordon Bell Prize competition occurred this year: A computer manufacturer was singled out for recognition. Nine entries reporting results obtained on the Cray C90 were received, seven of the submissions orchestrated by Cray Research. Although none of these entries showed sufficiently high performance to win outright, the judges were impressed by the breadth of applications that ran well on this machine, all nine running at more than a third of the peak performance of the machine.

Results

Lyle N. Long and Matt Kamon of Pennsylvania State University and Denny Dahl, Mark Bromley, Robert Lordi, Jacek Myczkowski, and Richard Shapiro of Thinking Machines were awarded $1,000 in the performance category for their model of a shock front. They used the more accurate, but more computationally demanding, Boltzmann equation rather than the approximate Navier-Stokes equation normally used. Their solution ran at more than 60 billion floating-point operations per second (Gflops) on a CM-5 with 1,024 processors.

Robert W. Means, Bret Wallach, and David Busby of HNC Inc. and Robert C. Lengel Jr. of Tracor Applied Sciences built a parallel system microcoded to be a signal processing computer that won them $500 for price/performance. Their machine, called SNAP (SIMD numerical array processor), enabled them to analyze images using the computationally demanding bispectrum analysis algorithm at 7.5 Gflops per million dollars.

Peter S. Lomdahl, Pablo Tamayo, Niels Gro/nbech-Jensen, and David M. Beazley of Los Alamos National Laboratory won $500 and honorable mention in the performance category for running an impressive 50 Gflops on a 1,024-node CM-5 simulating the microstructure of grain boundaries in solids. Equally impressive was the size of the problem: More than 100 million particles were used to model the cracking in a material being pulled apart.

Although they did not win a prize, Gary Sabot, Skef Wholey, Jonas Berlin, and Paul Oppenheimer of Thinking Machines were named finalists for automatically parallelizing a weather prediction model. The entire 65,000-line Fortran application was ported to a CM-5 using CMAX, a commercially available program that translates Fortran 77 into CM Fortran.

In an unusual move and a break from tradition in Gordon Bell Prize judging, Cray Research was named a finalist. Sara Graffunder, representing the company, summarized the work of nine teams who entered work performed on the new Cray C90. Entries came from four countries, and applications included molecular dynamics, seismic data processing, magnetohydrodynamics, fluid flow, structural mechanics, radar cross sections, and ab initio quantum chemistry.

The contrast between last year's winners and this year's was dramatic. This year the "big iron" from Thinking Machines and Cray dominated the workstation clusters, and performance rose sharply. Last year performance showed no improvement over previous years; this year

the best performing applications ran at rates almost eight times higher than last year's winning figures and almost five times higher than the previous record.

The machines

Two of this year's winning entries used Thinking Machines' CM-5, which can be configured with anywhere between 32 and 16,384 Sparc processors. Optionally, four vector units, each capable of 32 Mflops, can be added to each Sparc, giving the machine a peak performance of 2 Tflops (trillion floating-point operations per second). The winning entries both submitted runs made on 1,024-processor systems equipped with the optional vector execution units. Peak performance for this machine is 128 Gflops on 64-bit floating-point operands.

The CM-5 has three networks. The data network guarantees a 5-megabyte-per-second (MBps) data transfer rate simultaneously between pairs of nodes but can deliver up to 20 MBps in some circumstances. The control network is used for synchronization and reduction operations such as inner products. The third network is used to diagnose the system and is not accessible to the application programmer.

The previous offering from Thinking Machines, the CM-2, has a control processor that broadcasts a single instruction stream to the compute nodes, which operate in lockstep. On the CM-5, the control processor broadcasts a single program to all the nodes. Each node runs this program at its own rate, using the control network for fast synchronization. Synchronous communications, useful for data-parallel parts of a program, and asynchronous communications, useful for message-passing parts of the program, are both available via a library provided with the machine.

Each processor has four 8-Mbyte memory banks and can have four vector units, each connected to one of the banks. Memory is partitioned between the Sparc and the vector units. The Sparc can access the entire memory, but each vector unit can use data only in its own bank. Special instructions enable data to move between regions. Since the vector units can run at 32 Mflops and the memory can deliver only 16 million words per second to the vector units, peak performance can only be reached if data in the vector registers is heavily reused. These idiosyncrasies make the performance achieved by this year's winners all the more remarkable.

The Cray C90, used by nine entrants for a wide variety of applications, represents a more evolutionary change from the Y-MP than does the CM-5 from the CM-2. The largest machine, the C916, has 16 processors, each capable of running at 1 Gflops and having up to 1 billion words (a Gword) of memory. It can have up to 4 Gwords of solid-state disk, the slow memory accessed by special I/O routines. One significant C916 feature is its I/O capability, which provides up to 13.6 Gbytes per second aggregate bandwidth.

Performance winner

We have all been exposed to models of fluid flows. They are used on the nightly weather reports and to design the airplanes we fly in and the cars we drive. Designers of the America's Cup yachts use them, and they are even used to model the motion of droplets in ink jet printers. These models all share a common characteristic: The fluid is relatively dense. Since each particle in such a fluid travels only a small distance between collisions, we don't have to worry about the

motion of individual molecules. Thus, we can approximate the fluid as if it were continuous and model it with the Navier-Stokes equation.

Sometimes the density of a gas is so low that this approximation breaks down. As soon as the particles travel a distance equal to an appreciable fraction of the size of the objects being modeled between collisions, we have to use more complicated approaches. Two examples are the flow of gas over the space shuttle on reentry and the motion of a disk head over the platter.

Even the solution of the continuum problem in dense fluids is one of the Grand Challenges; the low-density case requires much more computation. The most commonly used approach is similar to the molecular dynamics calculations carried out by chemists modeling such things as protein folding. In the fluid case, we assign initial velocities and positions to a large number of particles. We can't include enough particles in the simulation to model the collisions directly. Instead, we take a Monte Carlo approach. At each time-step, some molecules are selected at random to collide with others. Each collision changes the velocities of the particles. These velocities are used to compute the new positions. This procedure is repeated thousands of times to produce averages that accurately represent the solution.

Since the molecules can interact only with others relatively nearby, the calculation is simplified by assigning each molecule to a cell. A molecule is allowed to collide only with another in the same cell. After each time-step, the assignment of particles to cells is updated. This is the step that makes parallelization hard, since it involves lots of interprocessor communication.

The entry submitted by Lyle N. Long, Matt Kamon, Denny Dahl, Mark Bromley, Robert Lordi, Jacek Myczkowski, and Richard Shapiro took a different approach by modeling the flow using the Boltzmann equation. We normally think of a model that keeps track of the distribution of the particles in three-dimensional space at each time-step. When solving the Boltzmann equation, we track the probability of finding a particle in a small volume of a six-dimensional region at each time-step. Three of the dimensions represent the positions of the particles; the other three represent the velocities. This position-velocity coordinate system is called phase space.

The Boltzmann equation has four terms. The first is the change of the particle distribution with time at each point in phase space. The second accounts for the flow of particles into and out of a region of phase space. Another term, which can be disregarded in the problems studied by this entry, accounts for external forces such as magnetic fields or gravity. The fourth is the most difficult to model. It represents the collisions between the particles and is a five-dimensional integral over velocity and scattering angle.

This integral is computationally intractable, so a number of approximations have been proposed. These approximations all share certain properties: They conserve mass, momentum, and energy; they give the proper solution in equilibrium (the Maxwell-Boltzmann distribution); and they force nonequilibrium solutions toward equilibrium. The approximation used in the calculations submitted by this entry has a particularly simple form that is valid near equilibrium: The collision term is the mean time between collisions multiplied by the deviation from equilibrium. Even with this simplification, the computational complexity is daunting, since we are computing a function with seven independent variables.

The solution scheme used discretizes the flow term with a finite-difference mesh and does the time-stepping with a Runge-Kutta method for the resulting ordinary differential equation. At each time-step, the physical properties of the gas -- density, velocity, and

temperature -- are calculated at each point in space by numerically integrating over the three velocity dimensions. A sufficiently accurate approximation can be obtained by combining values of the distribution at 343 points chosen according to the rules of Gaussian quadrature. One more step, a least-squares adjustment, is needed to ensure that conservation is maintained. This step involves a number of multiplications of 3 4 343 matrices by vectors with 343 elements.

This computation maps very well onto the CM-5. Each processor is assigned a point in physical space and holds the velocity information for that point. The time-step procedure involves communication because the flow term in the Boltzmann equation requires information from adjacent points in the mesh. However, both the integration over velocity to get the physical quantities and the correction term used to enforce conservation can be done without communication. Furthermore, the data in the vector registers is reused a lot, since the most time-consuming part of the calculation is the computation of the exponentials needed for the Boltzmann distribution. Each exponential takes 45 floating-point operations, more than enough to balance the limited memory bandwidth.

The Boltzmann equation method can be contrasted with the molecular dynamics approach for low-density problems. The molecular dynamics approach needs a small time-step, or the accuracy of the calculation suffers. In particular, the time-step must be short relative to the average time between collisions. Higher order methods, such as those that can be derived for the Boltzmann equation, can take much larger time-steps. In addition, the Monte Carlo molecular dynamics approach must repeat each time-step many times to get accurate averages to use as the solution. Combining the short time-steps with the number of repeated evaluations per time-step gives the Boltzmann equation approach a big advantage. For example, the molecular dynamics approach needed about 6,000 steps to solve the same problem that the Boltzmann equation method solved in five steps.

The problems submitted follow the evolution of a one-dimensional shock front and the boundary layer that forms as gas flows over a heated wall. While not important applications in their own right, they demonstrate that this extremely difficult computation is being done correctly and should be able to handle more realistic problems.

Straight compilation of the Fortran implementation ran at 40 Gflops -- more than 30 percent of peak performance. Examination of the code showed three places where the compiler did poorly. The compiler generated some unnecessary memory operations on a loop that multiplied a matrix times a vector. Recoding in assembly language improved the performance on this loop from 6 flops per 10 memory references to 24 flops per 16 references. Since most mature machine architectures have highly optimized library routines for this calculation, the use of assembly language is fully justified. A second loop was optimized to eliminate communications that end up moving data within a vector unit. The final optimization improved the handling of the boundary conditions. After these changes were made, the code ran at over 60 Gflops, almost half the peak speed of the CM-5.

Price/performance winner

You are a marine biologist following migrating whales. You are an astronomer listening to radio waves from space, trying to detect signs of intelligent life. You are responsible for maintaining a piece of equipment normally used 24 hours a day. You are a doctor studying an electro-encephalogram, looking for signs of small epileptic seizures in a patient. All four of these tasks share the need to detect a

signal in the midst of noisy data.

The whale watchers listen for the characteristic sounds whales make in the noisy ocean. The astronomer is trying to find a regular pattern mixed in with the random hissing from stars and galaxies. The equipment maintainer is trying to distinguish the characteristic sound a part makes as it wears out from the vibrations of normal operation. The doctor is looking for regular patterns of weak spikes among normal brain-activity waves.

Numerous schemes are used to solve these problems. Replica correlation assumes that I know what the signal looks like. I take the measured series and combine it with the expected, noise-free signal to produce a correlation value. A high correlation means that the signal is likely to be in the data. This scheme is computationally inexpensive, since only one multiplication and one addition are needed for each point in the sequence. However, I can't use it unless I know what the signal looks like.

Another frequently used procedure is to compute the power spectrum of the data. Each data sequence is transformed by taking the discrete Fourier transform to get a complex array. The power spectrum is the absolute values of these complex numbers. If the fast Fourier transform algorithm is used, only about five N log2 N operations are needed, so the method is quite efficient.

Conveniently, components with different frequencies produce distinct peaks in the power spectrum. If, for example, the data consists of a single sine wave, the power spectrum will be zero except for a single point at the corresponding frequency. If the data is white noise, the spectrum will be more or less flat with lots of wiggles, small bumps, and an occasional large bump.

The problem arises when noise is superimposed on several signals that are impure and periodic. The power spectrum will have broad peaks mixed in with the noise component. If the signal-to-noise ratio is small, the significant bumps will be hard to distinguish from the random bumps caused by the noise.

The power spectrum uses two independent values for each point: the real and the imaginary parts of the Fourier transform. Hence, it is a second-order spectrum. What if we combine more terms? We might be able to detect weaker signals. This is just what was done by Robert W. Means, Bret Wallach, David Busby, and Robert C. Lengel Jr., who combined three terms.

The third-order spectrum, usually called the bispectrum, has some interesting properties. First of all, white noise has a zero average bispectrum, which makes it easier to detect signals. More significantly, the bispectrum retains phase information lost when computing the power spectrum. Sometimes, as in the case of the electro-encephalogram, the phase information is important.

Unfortunately, the bispectrum is much more difficult to compute. The method used by our price/performance winners computes the discrete Fourier transform of the data, then forms the bispectrum by combining three terms, the Fourier coefficient at point j, the Fourier coefficient at point k, and the complex conjugate of the Fourier coefficient at point j + k. Thus, the computer time needed increases as the square of the number of points. Furthermore, each point in the bispectrum is a complex number, making it harder to interpret than the power spectrum.

Real-time performance is important in many areas that use the bispectrum. For example, doctors want to be notified when an epileptic seizure starts. Because of the large number of operations needed to compute the bispectrum, which is used to watch for the onset of a seizure, we would need a machine capable of supercomputer performance attached to each patient. Conventional supercomputers are too

expensive (even given the most ambitious proposals for universal health care), so a lower cost solution is needed. Our winners provide such a machine. They build and market the SNAP-32, a low-cost machine that can be microcoded to achieve supercomputer performance on certain problems.

A SNAP-32 has three components. A front-end host is used to control the system. A Balboa 860, a general-purpose computer with an Intel i860 processor and 16 to 64 Mbytes of memory, does the sequential part of the problem and loads the SNAP instruction unit with library microcode. The compute nodes, HNC100 chips, are configured four to a board, each chip consisting of four processors. The processors on a board are connected in a linear array. The SNAP-32 has two such boards connected to make a ring of processors. Each node has 512 Kbytes of memory. In addition, there is a 1.5-Mbyte global memory.

Each 20-MHz processor has a local memory and access to global memory. There is a floating-point multiplier and a unit that does floating-point addition and integer operations. Since the floating-point adder and multiplier can operate in parallel, the SNAP-32 has a peak performance of 1.3 Gflops.

A program is submitted from the front-end processor to the Balboa 860, which does all the sequential operations. When a parallel library routine is invoked, the Balboa transfers the microcode for the SNAP to the controller. In addition, the Balboa puts pointers to the function arguments into global memory. (The system knows when it is executing in a loop, and the Balboa will transfer the microcode only once.) The SNAP controller computes the addresses each node will need and sends them to the nodes. Once each node knows what part of the arrays to work on, the controller sends an operation, such as add or load, on every 50-nanosecond cycle.

The bispectrum is computed in four phases. First, the data is mapped to the local memories of the SNAP processors in a regular manner. Next, a fast Fourier transform (FFT) is done on the array. This step is followed by computation of the bispectrum. Finally, after many bispectra have been averaged, the data is converted to 8-bit images and transferred to the global memory, where it can be transferred to a display device.

In the problem submitted, FFTs were done on arrays 1,024 points in length. Each processor receives every 32nd element in the array, 32 elements per processor. An FFT is done on the 32 points in each processor independently of the others. The result is multiplied element by element by some trigonometric terms, and the ring interconnect is used to transpose the array. Next, the FFT is finished with another FFT on the new 32 points held by each processor. Finally, the data is transposed again to put the array back in natural order. The result, believe it or not, is the discrete Fourier transform of the original 1,024 data points.

Now we can compute the bispectrum from the discrete Fourier transform. This step is done by constructing a 512 4 512 matrix of complex numbers stored as 16 4 512 arrays in each processor's memory. We start with a vector of 16 elements held by each of the 32 processors. These elements are stored in the first row of each processor's part of the result matrix. Each processor sends its values to its left neighbor, which stores them in the second row of the matrix. These are passed left again and stored in the third row, and so on. After 32 shifts, each processor holds its part of the matrix. (We don't have to do all 512 shifts, since we can build 16 rows of the matrix after every shift. Also, symmetry is used to reduce the data stored to 256 4 512.) Next, we multiply each column of the matrix element-by-element with the computed FFT and the rows of this result element-by-element with the computed FFT. The end product is the

bispectrum of the original data.

The problem submitted computes 64 bispectra, averages them, and displays a gray-scale image on the Sun front end in 0.4 second; without the display, the SNAP-32 needs only 0.37 second. The same problem run on one processor of a Cray Y-MP/8 takes 0.7 second. Since the calculation does 150 million floating-point operations, the Cray is running at more than 210 Mflops, a sign of good optimization. The SNAP-32 is running at more than 400 Mflops, almost one third the peak performance of the machine. This performance is achieved on a system costing only $54,000, including the cost of the Sparcstation front-end machine. The 7.5 Gflops/million dollars achieved is more than four times the price/performance of the second-best entry this year.

Performance runner-up

A shoelace snaps. A tooth falls out of a comb. A balloon pops. These three minor tragedies of modern life share a common characteristic: The problem starts with a microscopic crack. The same mechanism is often responsible for such real tragedies as a bridge collapsing or a wing falling off an airplane. Given the importance of this problem, it is surprising how little we know about how such cracks start and propagate.

Does the crack start at a weak point on the edge and wend its way through the material following the path of least resistance? Do several small cracks start on the edge and spread until they join? What happens in the interior of the material? Does the temperature change in the vicinity of the crack? Some of these questions can be answered in the laboratory, but a deeper understanding of the underlying physics may come only from detailed simulations.

Computer simulations of bulk materials have been carried out using a few million atoms. In tens of hours on conventional supercomputers, these models follow the material for a few nanoseconds. Unfortunately, the materials being studied are dense enough that a billion atoms fill a cube only a third of a micron on a side. (Today's most advanced semiconductor circuits use elements about a third of a micron across.) In addition, we would like to follow the material for a second or so, but even a few nanoseconds reveals important physics. Since the largest time-steps a simulation can take are only about 10 femtoseconds (10 millionths of a nanosecond), we need to compute tens of millions of time-steps. Only when we can solve problems at least this big can we begin to compare the computer models with experiments on macroscopic samples.

Such simulations are conceptually simple. Compute the force of every atom on every other atom, then use these forces to compute the accelerations of each atom. Next, compute the new velocities and new positions. Repeat the process a few million times and you're done. The difficulty is the sheer size of the problem. If I want to follow the motion of 1 billion atoms, I will have to compute about 1 billion billion forces. If I could complete a force calculation in 1 microsecond, each time-step would take about 15,000 years! Clearly, I need a way to reduce the number of forces computed.

One of last year's Gordon Bell Prize winners modeled galaxies with millions of stars. Since the gravitational force of each star extends to infinity, all stars in the galaxy had to be included in the force calculation. Studying neutral atoms in bulk materials is somewhat easier because the force is appreciable only a small distance from each atom. In a typical run, each particle is affected by only a few hundred other atoms. Furthermore, the number of atoms that must be included in calculating the force on a given atom depends only on the

density of the material being studied, not on the number of atoms in the simulation. The trick is finding out which atoms to include in the force calculation.

Molecular dynamics calculations typically use two data structures to find the neighbors of each atom. One is simply a list with one entry for each atom containing its position, velocity, and physical properties. The other is a discretization of the spacial domain being modeled. Space is divided into cubes just a little larger than the distance over which the molecular forces can be felt. Each atom lies in a cell and can be affected only by atoms in the same or neighboring cells.

The algorithm is now straightforward. At each time-step, use the position of each atom to assign it to a cell. Compute the forces on each particle from all the others in the same cell and the neighboring cells. Update the positions and velocities, and repeat over and over. If it takes only 1 microsecond to compute the force between two atoms, a simulation with 1 billion atoms can be done in only 30 minutes of computer time per time-step using this method. Hence, every two days I can advance the simulation time about 1 picosecond (a thousandth of a nanosecond).

The problem now is how to compute the force on an atom in a microsecond. It takes about 50 floating-point operations to compute the force between a pair of atoms. At 25 Mflops, this calculation takes 2 microseconds. Since each atom interacts with as many as 500 other atoms, the force calculation would take about 1 millisecond per particle. Even a conventional supercomputer running at 250 Mflops would take 100 microseconds to compute the force on an atom.

The solution taken in the entry submitted by Peter S. Lomdahl, Pablo Tamayo, Niels Gro/nbech-Jensen, and David M. Beazley was to use 1,000 processors to do the computation. Since each processor needs to compute the force on only 1 million atoms, the update time per particle can be reduced to the microsecond range. Even with this high degree of parallelism, the entry takes only a small step toward achieving the goal of modeling macroscopic materials by following the motion of individual atoms.

Two difficulties must be addressed in mapping the algorithm to the CM-5. First, each processor will hold a subset of the cells and the atoms located in those cells. Some of the cells will have neighbors held by other processors, so there will be communication during some of the force calculations. The performance of this part of the code is improved by keeping the particle data for each cell together in memory to make it easier to transfer. Furthermore, after each time-step, some of the particles will have moved from one cell to another, and some will have moved to a neighboring processor. Little can be done to improve this part of the program. In fact, communication accounts for less than 10 percent of the time in the largest runs because each processor has so many cells that the fraction of the cells needing to communicate with another processor is small.

Another difficulty is figuring out how to use the vector units effectively. Remember that the vector units perform at close to their peak rates only if data can be reused several times. Reuse is increased by using knowledge of the force calculation. Look at the force on an atom in cell 1 resulting from all the atoms in cell 2. Load up the vector registers with data for 32 atoms in cell 2. Now, compute the force on each atom in cell 1 that results from these 32 atoms. Repeat for the next 32 atoms in cell 2. Since each cell contains many atoms, the data loaded into the vector registers is used many times. A further improvement comes from optimizing the movement of data between the Sparc and vector portions of the memory.

These optimizations led to impressive performance. The model

submitted for the competition executed 25 time-steps at more than 50 Gflops following the evolution of 100 million particles. Models with as many as 11 million particles have been run for up to 14 hours. (The larger run would have used all the memory on the CM-5, irritating others wishing to use the machine.) Some people will be surprised to learn that this difficult calculation can run at 40 percent of the theoretical peak performance of the machine.

Unfortunately, we will have to wait for the next generation of systems to make comparisons with -- and perhaps improve on -- laboratory data. However, we are getting close. Some experiments being done with femtogram particles can be modeled with only 10 times the particles that can be handled today.

Other finalists

Gary Sabot, Skef Wholey, Jonas Berlin, and Paul Oppenheimer automatically parallelized a production weather code, achieving a speedup of more than 900 on a 1,024-processor CM-5. The Advanced Regional Prediction System (ARPS) code is used by many groups, including the Center for Analysis and Prediction of Storms at the University of Oklahoma, to predict the formation of dangerous weather patterns such as tornadoes. Since time is of the essence, the calculation should run faster than the storm develops.

This team ran the entire 65,000-line ARPS Fortran program through the CMAX translator, a product from Thinking Machines that translates Fortran 77 into CM Fortran. They started from a good base, since ARPS was designed with modern vector and parallel machines in mind. Even so, simply porting the application involved inserting some 65 compiler directives to specify the data layout and the absence of dependences in some loops. Five other optimizations commonly done when moving a program to a new machine, such as in-lining some short subroutines, improved the performance by about a factor of three. These changes made it possible for the compiler to generate code that ran at about 9 Gflops and simulate one hour's worth of weather in just under five hours of computer time. With only one or two more orders of magnitude in performance, it will be possible to model the weather fast enough to warn of tornadoes on the way.

Nine of the entries reported results achieved on the latest Cray Research offering, the C90, leading to special recognition for Cray. Performance ranged from one third to two thirds of the 16-Gflops theoretical peak rate of the machine. The range of applications achieving such high performance was impressive, covering topics from seismic migration to molecule folding to magnetohydrodynamics to fluid flows to boundary integrals. Particularly interesting was the 6 Gflops achieved running the commercial structural-analysis code Ansys.

Other entries

Ten entrants submitted impressive work but did not make it to the finals. The entry with the highest performance overall reported 64 Gflops on a seismic analysis program running on a 1,024-processor CM-5. Only one entry was received for work done on an Intel parallel machine; it reported simulation of the quantum mechanical behavior of a high-temperature superconductor on a 128-node Intel Delta at a rate some 30 times faster than that achieved on a Cray Y-MP processor.

The only cluster-computing entries were for an analysis of DNA sequences and two applications run on 30 Sun workstations using Network Linda. In addition to the SNAP, SIMD (single instruction,

multiple data) computing was represented by entries run on the Maspar MP-1 and MP-2 to simulate molecular dynamics, predict the folding pattern of a molecule of RNA, and train neural networks to retrieve atmospheric parameters from infrared spectra. Two other entrants submitted fluid-flow calculations on the CM-5.

The judges

Alan H. Karp, who chaired the judging committee, is a senior member of the technical staff at Hewlett-Packard Laboratories in Palo Alto, California.

Don Heller was, until recently, on the staff of Shell Oil Company's Computer Sciences Research Department. He is currently at the Center for Research on Parallel Computation at Rice University.

Horst Simon is a staff member at Computer Sciences Corporation working under NASA Contract NAS 2-12961 at the Ames Research Center in Moffett Field, California.

Sidebar

Seventh annual Gordon Bell Prize

The Gordon Bell Prize recognizes achievements in large-scale scientific computing. The rules for the prize will not change in 1994. Entries are due on May 1, 1994, and finalists will be announced by June 30. Pending approval by the Supercomputing 94 program committee, finalists will be invited to present their work at a special session of that conference in November. Winners and honorable mentions will be announced following the presentations.

The 1994 prizes will be given for work in the following categories:

Performance: The entrant will be expected to convince the judges that the submitted program is running faster than any other comparable engineering or scientific application. Suitable evidence will be the megaflop rate based on actual operation counts or the solution of the same problem with properly tuned code on a machine of known performance, such as a Cray Y-MP. If neither of these measurements can be made, the submitter should document the performance claims as well as possible.

Price/performance: The entrant must show that the performance of the application divided by the list price of the smallest system needed to achieve the reported performance is better than that of any other entry. Performance measurements will be evaluated as for the performance prize. Only the cost of the CPUs, memory, and any peripherals critical to the application need be included in the price. For example, if the job can be run on diskless compute servers, the cost of disks, keyboards, and displays need not be included.

Compiler parallelization: The combination of compiler and application that generates the greatest speedup will be the winner. Speedup will be measured by dividing the wall clock time of the parallel run by that of a good serial implementation of the same job. These runs may be the same program if the entrant can convince the judges that the serial code is a good choice for a uniprocessor. Compiler directives and new languages are permitted. However, anyone submitting an entry in other than a standard, sequential language will have to convince the judges that the parallelism was detected by the compiler, not by the programmer.

Some general conditions apply:

(1) The submitted program must have utility; it must solve a problem that is considered a routine production run, such as making daily weather predictions or solving an important engineering or scientific problem. It should not be a contrived or experimental problem that is intended just to show high speedup.
(2) Entrants in the price/performance category must demonstrate that the machine they used has real utility. (Picking up a few used Z-80s for $1 each is not acceptable.) Only list prices of components should be used. If the machine is not on the market, the entry is probably not eligible, although the judges will consider any reasonable estimate of the price.
(3) One criterion the judges will use for all categories is how much the entry advances the state of the art in some field. For example, an entry that runs at 120 Gflops but solves a problem in a day that previously took a year might win over an entry that runs at 150 Gflops solving a more mundane problem. Entrants who believe their submission meets this criterion are advised to document their claims carefully.
(4) In all cases the burden of proof is on the contestants. The judges will make an honest effort to compare the results of different programs solving different problems running on different machines, but they will depend primarily on the submitted material.

Contestants should send a three- or four-page executive summary to Marilyn Potes, IEEE Computer Society, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1264 before May 1, 1994.